Code Clones

SW Maintenance and Evolution

Emanuel Giger
Code Clones are similar segments of source code found in different places of a system.
Two Fundamental Questions?

How do we define similar? When do we consider code segments to be similar?

How do we detect similar source code segments?
Two Fundamental Questions?

Both problems are surprisingly difficult

Find *all* clones but avoid *false positives*

*Fast* and *efficient*: Software systems are potentially large
1. Exact Copies

```c
if (m_swapChain)
    m_swapChain->SetFullscreenState(false, NULL);

if (m_rasterState)
    m_rasterState->Release();
    m_rasterState = 0;

if (m_depthStencilView)
    m_depthStencilView->Release();
    m_depthStencilView = 0;

if (m_depthStencilState)
    m_depthStencilState->Release();
    m_depthStencilState = 0;

if (m_depthStencilBuffer)
    m_depthStencilBuffer->Release();
    m_depthStencilBuffer = 0;

if (m_renderTargetView)
    m_renderTargetView->Release();
    m_renderTargetView = 0;
```
2. Parameterized Copies

// Set up the description of the static vertex buffer.
vertexBufferDesc.Usage = D3D11_USAGE_DEFAULT;
vertexBufferDesc.ByteWidth = \texttt{sizeof(\texttt{Vertex\_Type})} \times m_{\text{vertexCount}};
vertexBufferDesc.BindFlags = D3D11_BIND_VERTEX_BUFFER;

// Give the subresource structure a pointer to the vertex data.
vertexData.pSysMem = \texttt{vertices};
vertexData.SysMemPitch = 0;
vertexData.SysMemSlicePitch = 0;

// Now create the vertex buffer.
result = device->CreateBuffer(&vertexBufferDesc, &vertexData, &m_{\text{vertexBuffer}});

// Set up the description of the static index buffer.
indexBufferDesc.Usage = D3D11_USAGE_DEFAULT;
indexBufferDesc.ByteWidth = \texttt{sizeof(\texttt{unsigned\_long})} \times m_{\text{indexCount}};
indexBufferDesc.BindFlags = D3D11_BIND_INDEX_BUFFER;

// Give the subresource structure a pointer to the index data.
indexData.pSysMem = \texttt{indices};
indexData.SysMemPitch = 0;
indexData.SysMemSlicePitch = 0;

// Create the index buffer.
result = device->CreateBuffer(&indexBufferDesc, &indexData, &m_{\text{indexBuffer}});
3. Extended Copies

```java
while (unsorted) {
    unsorted = false;
    for (int i = 0; i < x.length - 1; i++)
        if (x[i] > x[i + 1]) {
            temp = x[i];
            x[i] = x[i + 1];
            x[i + 1] = temp;
            unsorted = true;
        }
}
```

```java
while (unsorted) {
    unsorted = false;
    for (int i = 0; i < x.length - 1; i++)
        if (x[i] > x[i + 1]) {
            try{
                temp = x[i];
                x[i] = x[i + 1];
                x[i + 1] = temp;
            } catch (IndexOutOfBoundsException) {
                ErrorLogger.("...");
                return false;
            }
            unsorted = true;
        }
}
```
Code Clones Types

Cloned code segments can be found in *different files*, in the *same files* but in *different methods*, or in the *same method*

Segments must contain some *kind of logic or structure* that can be abstracted

....
computeVectorCrossProduct(v1, v2);
....

....
computeVectorCrossProduct(v2, v3);
....

....
setIP("182.89.34.21");
....

....
setIP("182.89.34.21");
....
Cloned code segments can be found in **different files**, in the **same files** but in **different methods**, or in the **same method**.

Segments must contain some **kind of logic** or **structure** that can be abstracted.

Most likely not a clone.

```java
computeVectorCrossProduct(v1, v2);
...
setIP("182.89.34.21");
...
computeVectorCrossProduct(v2, v3);
...
setIP("182.89.34.21");
...```
Cloned code segments segments can be found in *different files*, in the *same files* but in *different methods*, or in the *same method*

Segments must contain some *kind of logic or structure* that can be abstracted

```java
.....
computeVectorCrossProduct(v1, v2);
....

Most likely not a clone
.....
computeVectorCrossProduct(v2, v3);
....

Potential clone
.....
setIP("182.89.34.21");
....
setIP("182.89.34.21");
....
```
Copied artifacts range from expressions, to functions, to data structures, and to entire subsystems.
Why Code Clones?

- Ctrl&C and Ctrl&V: Simple and fast way of code reuse, “templating”
- No time to factor out useful code
- Unexperienced developers
- Technological constraints: Frameworks, programming language (e.g., no polymorphism)
- Architectural constraints
- ...

Bad Clones

- Code bloat
- Doubles, triples, quadruples, ... the effort maintenance: *Code reading* and *modification*
- Copied defects
- Risk of *inconsistent changes* or “forgotten” clone segments
- Increases testing efforts when clones are scattered throughout different files/methods
Simply, it is a question regarding the aesthetic of code.
• 8 to 12% in normal industrial code

• gcc: 8.7%, LOC 460k

• Database Server: 36.4%, LOC 245k

• X Windows: 19%

• Payroll Software: 59.3%

• Mostly small clones < 25 LOC

How much Code Clones are out there?
Active Research Topic

- Code Clone Detection
- Consequences of clones to maintenance
- Tool support
- Visualization of Code Clones
- Characteristics of Code Clones
Clones considered harmful? - Studies are inconclusive

Cory Cooper and Michael W. Godfrey
Software Architecture Group (SWAG)
David R. Cheriton School of Computer Science, University of Waterloo
{cj cooper, mgodfrey}@uwaterloo.ca

Abstract

Current literature on the topic of duplicated (cloned) code in software systems often concludes that code clones are considered harmful. However, we found that the evidence for this conclusion is often based on case studies that may not be conclusive, and there are studies that have found cloning to be beneficial. In this paper, we present a novel algorithm for detecting code clones that can be applied to a variety of software systems. We then use this algorithm to conduct an empirical study of code cloning in several large software systems. Our results suggest that code cloning is common in software systems, but that its harmfulness is often overstated. We recommend that further research be conducted to better understand the impact of code cloning on software systems.

1. Introduction

We believe that most large-scale systems contain a non-trivial amount of redundant code. Often related to cloning, these segments of code typically account 10-15% of the source code. [24, 25]. Code clones can arise through various mechanisms. For example, intentional clones may be introduced through direct "copy-and-paste" of code. Unintentional clones on the other hand may be introduced through the language or the evolving design of the system. [1, 2, 20].

2. Clones & correctness

In software maintenance, a clone is a source code fragment whose structure is identical to another fragment of source code. Clones are considered harmful because of the potential for inconsistent changes to the clone. [1, 2, 20]. A clone is a maximal set of source code fragments that are identical to each other. [24, 25]. A clone class is a maximal set of source code fragments that have the same structure. [24, 25].

3. Clones & maintenance effort

Maintenance effort on methods is important to identify when a clone is present. [24, 25]. Maintenance effort on methods seems to increase depending on the percentage of source code that is cloned. [24, 25].

4. Clones & correctness

Although most previous research agrees that code cloning poses a problem for software maintenance, there is little internal information available concerning the impact of code clones on software quality. [24, 25]. This is particularly true in the context of code clones that have cause faulty behavior. [24, 25]. Clones seem to increase the effort needed to fix bugs in code. [24, 25].

5. Clones & maintenance effort

Maintenance effort on methods is important to identify when a clone is present. [24, 25]. Maintenance effort on methods seems to increase depending on the percentage of source code that is cloned. [24, 25].

6. Clones & correctness

Although most previous research agrees that code cloning poses a problem for software maintenance, there is little internal information available concerning the impact of code clones on software quality. [24, 25]. This is particularly true in the context of code clones that have cause faulty behavior. [24, 25]. Clones seem to increase the effort needed to fix bugs in code. [24, 25].

7. Clones & maintenance effort

Maintenance effort on methods is important to identify when a clone is present. [24, 25]. Maintenance effort on methods seems to increase depending on the percentage of source code that is cloned. [24, 25].
Non-trivial Problem: *No a priori* knowledge about which code was cloned, its amount, its granularity.

How to find all cloned segments among all possible pairs of segments: *Avoid computational complexity*.

Define an appropriate *similarity measure*: Some clones are similar, but not identical - abstract from those differences.
Code Clone Detection

Transformation

Source Code → Transformed Code → Duplication Data

Comparison
# Code Clone Detection

<table>
<thead>
<tr>
<th>Author</th>
<th>Level</th>
<th>Transformed Code</th>
<th>Comparison Technique</th>
</tr>
</thead>
<tbody>
<tr>
<td>Johnsen, 1994</td>
<td>Lexical</td>
<td>Substrings</td>
<td>String-Matching</td>
</tr>
<tr>
<td>Ducasse, 1999</td>
<td>Lexical</td>
<td>Normalized Strings</td>
<td>String-Matching</td>
</tr>
<tr>
<td>Baker, 1997</td>
<td>Syntactical</td>
<td>Token Strings</td>
<td>String-Matching</td>
</tr>
<tr>
<td>Mayrand, 1996</td>
<td>Syntactical</td>
<td>Metric Tuples</td>
<td>Discrete comparison</td>
</tr>
<tr>
<td>Kontogiannis, 1997</td>
<td>Syntactical</td>
<td>Metric Tuples</td>
<td>Euclidean distance</td>
</tr>
<tr>
<td>Baxter, 1998</td>
<td>Syntactical</td>
<td>AST-Representation</td>
<td>Tree-Matching</td>
</tr>
</tbody>
</table>
Code is *tokenized* by set of lexical rules (Lexical Analysis)
Token Based Clone Detection

Code is *tokenized* by set of lexical rules (Lexical Analysis)

Token sequence is *transformed/normalized* by a set of rules
Rules Examples:

C++ Rules:

Remove namespace: `std::vector` to `vector`

Remove template parameters: `vector<int>` to `vector`

Java Rules:

Remove modifiers: `protected void foo()` to `foo()`
Token Based Clone Detection

Code is *tokenized* by set of lexical rules (Lexical Analysis)

Token sequence is *transformed/normalized* by a set of rules
Token Based Clone Detection

Code is *tokenized* by set of lexical rules (Lexical Analysis)

Token sequence is *transformed/normalized* by a set of rules

Identifiers of variables, types, methods, and constants are *replaced with a special token*

Code segments with different variable names become clone pairs
Token Based Clone Detection

Code is tokenized by a set of lexical rules (Lexical Analysis)

Token sequence is transformed/normalized by a set of rules

Detection based on the tokenized source code

Identifiers of variables, types, methods, and constants are replaced with a special token

Code segments with different variable names become clone pairs
Token Based Clone Detection
Token Based Clone Detection

```cpp
void print_lines(const set<string>& s) {
    int c = 0;
    set<string>::const_iterator i = s.begin();
    for (; i != s.end(); ++i) {
        cout << c << " , "
             << *i << endl;
        ++c;
    }
}
```
void print_lines (const set & s) {
    int c = 0;
    const_iterator i = s.begin();
    for (; i != s.end(); ++i) {
        cout << c << "," << *i << endl;
        ++c;
    }
}

void print_lines ( const set & s ){
    int c = 0;
    const_iterator I = s.begin();
    for (; i != s.end(); ++i) {
        cout << c << "," << * I << endl;
        ++ c;
    }
}
Token Based Clone Detection

```cpp
void print_lines ( const set & s )
{
    int c = 0;
    set::const_iterator i = s.begin();
    for (; i != s.end(); ++i) {
        cout << c << ", "
             << *i << endl;
        ++c;
    }
}

void print_lines ( const set & s )
{
    int c = 0;
    Const_iterator I = s.begin();
    for (; i != s.end(); ++i) {
        cout << c << ", "
             << *I << endl;
        ++c;
    }
}

$p  \$p ( \$p  \$p  &  \$p )$
$p  \$p  =  \$p ;$
$p  \$p$
    = \$p  .  \$p ( ) ;$
for ( ; \$p  !=  \$p  .  \$p ( ) ;  ++  \$p )$
{
    \$p  \$p
    \$p  \$p
    \$p  \$p
    \$p
    \$p  \$p
    \$p
    \$p$
}
Basic idea: For a given code segment a metrics profile is calculated.

Basic assumption: Similar code segments have similar metrics profiles.

Granularity: Method/Function level.

Approach makes sense: Most segments are copied & pasted (and slightly modified).

But they keep their basic properties.
Metrics Based Clone Detection Mayrand’96

- 4 Points of comparison:
  - Name
  - Layout
  - Expressions
  - Control flow

- Each point is a set of numerical metrics describing certain aspects of a method/function
“If two functions have the same name they are likely clones.”

Relative number of common characters
2. Point of comparison: Layout

Layout is defined as: “the visual organization of the source code”

Count the number of comments, number of non-blank lines, average length of variable names, ...
3. Point of comparison: Expressions

“the number of expressions in a function, their nature and their complexity are considered”

Count calls to other methods, number of declaration statements, number of executable statements, conditional complexity
4. Point of comparison: Control Flow

"the control flow characteristics of a method"

Number of unique paths, number of loops, nesting level, number of exits, number of conditional decisions, ...
<table>
<thead>
<tr>
<th>Scale</th>
<th>Nam</th>
<th>Lay</th>
<th>Exp</th>
<th>Con</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-ExactCopy</td>
<td>=</td>
<td>=</td>
<td>=</td>
<td>=</td>
</tr>
<tr>
<td>2-DistinctName</td>
<td>!=</td>
<td>=</td>
<td>=</td>
<td>=</td>
</tr>
<tr>
<td>3-SimilarLayout</td>
<td>X</td>
<td>~</td>
<td>=</td>
<td>=</td>
</tr>
<tr>
<td>4-DistinctLayout</td>
<td>X</td>
<td>!=</td>
<td>=</td>
<td>=</td>
</tr>
<tr>
<td>5-SimilarExpression</td>
<td>X</td>
<td>X</td>
<td>~</td>
<td>=</td>
</tr>
<tr>
<td>6-DistinctExpression</td>
<td>X</td>
<td>X</td>
<td>!=</td>
<td>=</td>
</tr>
<tr>
<td>7-SimilarControlFlow</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>~</td>
</tr>
<tr>
<td>8-DistinctControlFlow</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>!=</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>=</td>
<td>Equal values for all metrics in a point of comparison between two functions</td>
</tr>
<tr>
<td>~</td>
<td>At least one metric not equal but within the delta in a point of comparison between two functions</td>
</tr>
<tr>
<td>!=</td>
<td>At least one metric not equal and outside the delta in a point of comparison between two functions</td>
</tr>
<tr>
<td>X</td>
<td>The point of comparison is not considered in the scale evaluation.</td>
</tr>
</tbody>
</table>
Clone Visualization

Software systems in practice are large thousands of files.

Cloned code exists in different files, different methods of the same files.

How can we present the results of clone detection?

Quick and efficient overview where similarities in source code occur.
Dot Plot Visualization

Adopt idea for DNA Analysis: Compare protein and DNA sequences

Dot plot is an established technique to compare sequences

(not to be confused with dot plots from statistics)
<table>
<thead>
<tr>
<th>T</th>
<th>G</th>
<th>A</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>G</td>
<td>A</td>
<td>G</td>
<td>G</td>
</tr>
<tr>
<td>G</td>
<td>G</td>
<td>C</td>
<td>T</td>
</tr>
</tbody>
</table>

A

A

T

A

A

T

C

G

G

C

A

T
(No sequence length restrictions)
(Clone sequence length > 2)
Dot Plot Visualization

- Code (according to granularity) is put on vertical / horizontal axis
- A match between two elements is a dot in the matrix
- Easy visual identification of insertion, deletions, repeats, variations

![Dot Plot Examples]

Exact Copies

Copies with Variations

Inserts/Deletes

Repetitive Code Elements

Image by Uni of Berne [Rieger & Ducasse 1999]
Example: Copied Code Sequences

- File A contains two copies of a code segment
- File B also contains two copies of that segment
- Extract Method to refactor clone?
- Examples are made using Duploc from an industrial case study (1 Mio. LOC C++ System)
- Duploc @ Uni Berne [Rieger & Ducasse 1999]
Example: Cloned Class

One Class is an edited copy of another class

Subclassing to refactor clones?
Dot Plots

+ Good overall impression
+ Easy to spot patterns
- Pairwise comparison
- Scalability?
• Code Clones exist and can be problematic during maintenance (inconclusive results from research!)

• Solution: Periodic clone assessment of a software, e.g., major releases,

• Detection of clones is nontrivial

• Efficient visualization of code clones is needed for real world system

• There is tool support